Software Security for the People: Free and Open Resources for Software Security Training

Article *in* IEEE Security and Privacy Magazine · March 2022

DOI: 10.1109/MSEC.2022.3142336

Barton P. Miller
University of Wisconsin–Madison
238 PUBLICATIONS **9,228** CITATIONS

Software Security for the People: Free and Open Resources for Software Security Training

Elisa R. Heymann
University of Wisconsin-Madison

Barton P. Miller
University of Wisconsin-Madison

***Abstract*—The enormous growth in software development affects every facet of our lives, creating an urgent need for training in software security. In response, we have developed free and open software security education and training materials for a wide range of practitioners, from the student to experienced professional.**

■ INDUSTRY, GOVERNMENT, AND ACADEMIA are developing software across a wide range of critical areas, including online services, sensor networks, autonomous vehicles, and IoT where computing is found in a myriad of physical devices. The confidentiality, integrity, and availability of this software and devices that are controlled by it is an ongoing concern. Software developers need the skills to design, write, test, and assess code that is resistant to being exploited [1,2]. However successful exploits appear at a frightening rate.

We are addressing these needs by:

1. Developing and deploying a curriculum and instructional and training materials to help software practitioners design, write, and assess code that is resistant to attack. These materials are based on our experiences in doing in depth vulnerability assessments of critical systems.
2. Producing curriculum and training materials that are cross-cutting and can readily be applied to undergraduate and graduate education, and to the training of software professionals.
3. Applying a modular approach to our curricula and training materials so that they vary in depth and length, cover a variety of security attacks, and provide examples based on different operating systems, programming languages, and frameworks.
4. Providing materials and instruction/training in a scalable way so that we can reach the broadest audience. Scalability is addressed in several ways, including: available online video lectures, text chapters, and exercises; producing an advanced undergraduate course at our university; live training at conferences, labs, and universities; turnkey teaching materials for the use of other instructors; week-long software security "boot

camps" for undergraduates from institutions that serve underrepresented communities; and train-the-trainers workshop to increase the number of qualified software security instructors.

5. Developing student and trainee evaluation and assessment materials, with the ultimate goal of providing levels of certification.
6. Focusing on accessibility by providing closed captions on all our video materials. Initially, these captions are available in both English and Spanish.
7. Lowering the barriers to acceptance and use of these materials by making them free and open.
8. Ensuring that our materials are a living curriculum that can be updated to match the frantic technological developments in cybersecurity. Leveraging the free and open nature of our materials, we are developing an open source community around them. We currently have colleagues developing new modules in areas in which they have special experience.

While we have made much progress in producing materials and using them in education and training, this effort is a living work with new developments yet to be made.

THE NEED

There is an urgent need for software practitioners that are better trained (or trained at all) in software security. There is broad competition across many industries for practitioners with software security skills. Industries in areas that are not traditionally involved in computing are now competing for new graduates and experienced practitioners. A better supply of software-security trained new graduates *and* educational resources are needed for our existing workforce. We also need to reach out to underrepresented groups to help build an inclusive workforce.

Looking beyond the dramatic headlines, we can see concrete numbers that motivate the urgency for better resources in the area of software security. The last decade has brought an incredible proliferation of software, with a rate of growth unseen in any previous time. The most dramatic sources of this growth are application stores for both the iOS and Android, where there are now millions of apps in each store (see Figure 1). We can also see this growth in the proliferation of web sites with active content; in control systems for trains, cars, aircraft, and shipping; and in the increasing appearance of embedded computers in the home, workplace, and even the objects that we wear or carry with us (or inside of us).

In this age of unprecedented growth of software, we are producing more programmers now than we did at the peak of the decades-ago Internet Bubble, but not nearly in proportion to the growth in software production (see the undergraduate degree data from CRA Taulbee Survey in Figure 2). The number of degree recipients in CS and CE is not matching the demand for such students and few of these graduates are trained in software security [3]. We have observed the same lack of training in software security for experienced industrial and research software teams. For other programmers – those informally trained or self-taught, or those with limited software academy experience – security seems to be a distant and vague concept.

In many organizations, the concept of software security is poorly understood and often poorly funded [3]. Most organizations understand the networking and hardware aspects of security, the firewalls, border routers, web and email filters, virus scanners, and perhaps even identity management; however, many of these organizations deploy services and build devices based on software that they write, either standalone or as part of a web infrastructure. The security of this code, from the design, to the coding, to the testing, to the deployment, is often an afterthought, if it is considered at all.

APPROACH

Our approach is to address software security education in a holistic way, by developing common materials and curricula that apply to a broad spectrum of learning contexts. We include video units with closed captions in English and Spanish, accompanying text chapters, presentation materials, assessment materials, and exercises. The goal of this project is to reach the broadest audience possible in an area where there is a desperate shortage of trained practitioners.

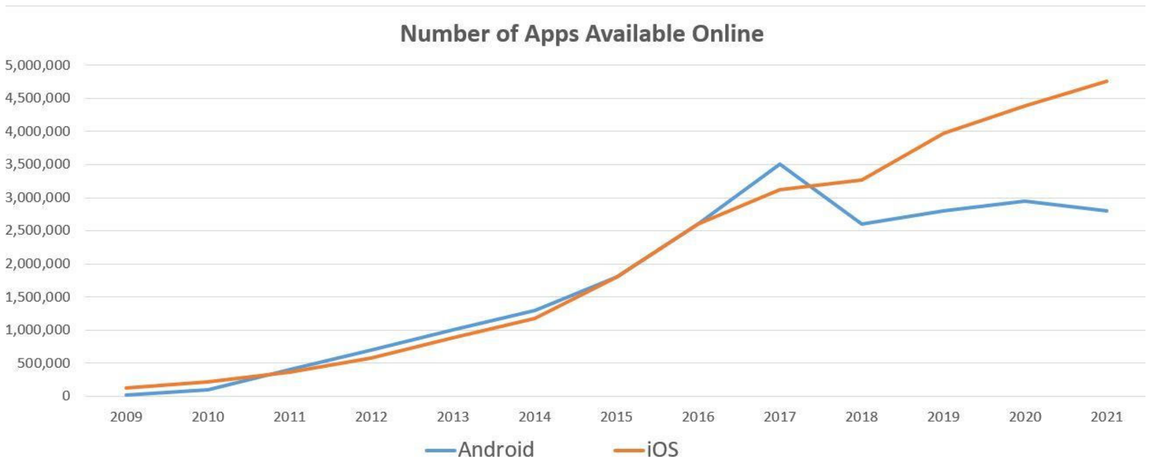


Figure 1: Number of Mobile Apps for Android and iOS

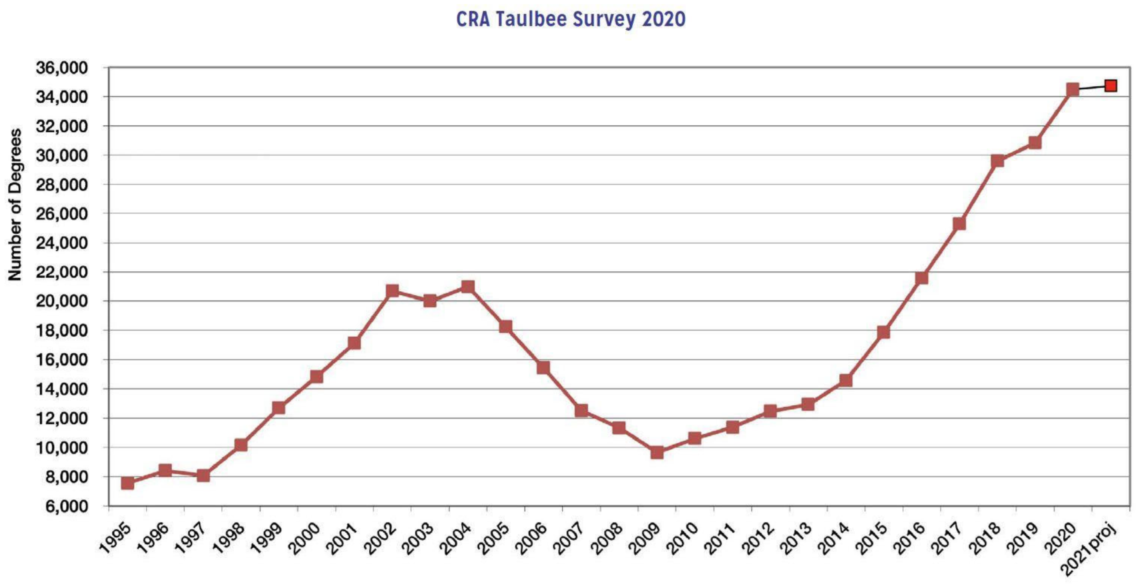


Figure 2: U.S. Undergraduate CS and CE Degree Production [4]

Much of this work has been supported TrustedCI, the NSF Cybersecurity Center of Excellence. Our role in that project has been to conduct in-depth software vulnerability assessments of scientific infrastructure projects, develop software security training materials, and conduct live training related to software security.

Guiding Principles

Our approach to curriculum and materials is based on several guiding principles:

P1. Span technical depth:

Our learning materials are targeted to span the range from students and trainees to software professionals with little security experience and experienced software professionals who want to further develop their security skills. Our materials support starting points from a basic programming background to experienced practitioners and advanced graduate students. By using a small-unit structure and presenting materials with increasing opportunities for depth, we can reach students and trainees at their current skill level. As they increase their basic programming and computer science knowledge, they can access more units and more depth in previously-studied units.

P2. Span technical breadth:

The software security field has many dimensions. For example, there are many technical areas, many commonly used languages, and several commonly used operating systems and platforms. While no curriculum can be truly universal, we are building it in such a way that it has broad applicability. Some areas, such as pointers and memory, are targeted at specific languages (C and C++ in this case). Some areas, such as exception handling, are applicable to many languages, and benefit from examples or detailed mention in each language. Other areas, such as techniques for fuzz testing or vulnerability assessment, are agnostic to the language and platform. As a result, the selection of units, sections from a selection of units, or focus on a language or platform in the units, will allow the instructor or students and trainees to target their learning.

P3. Accessibility and inclusivity:

It is crucial to reach a broad audience across cultural boundaries. The first step in that process will be to produce closed captions for each video unit. We currently include both English and Spanish captions. As opportunities present themselves, we will add other languages (we already have informal offers from colleagues in Japan and Germany). We try to conform to Web Content Accessibility Guidelines (WCAG) to ensure that we create the fewest barriers possible to our materials. In addition, we continue to solicit feedback from our users to help us be more inclusive.

P4. Broad reach:

There are many opportunities to introduce software security, from the start of a programmer's education, to a comprehensive university software security course for a computer science major, to focused training for the practicing software professional. The above principles of spanning technical breadth and depth support our ability to have broad reach.

P5. Clear learning goals:

Our curriculum is formulated to have clear learning outcomes for each unit. For the student, they see a concrete list of goals presented at the start of each video lecture. For the instructor, there will be an expanded list of learning outcomes, prerequisites, and evaluation methods for each unit.

P6. Shared materials:

We have created widely accessible delivery channels to ensure that our materials are broadly available and freely accessible. These channels include a university-supported website for text chapters and hands-on exercises, and Vimeo for the videos (see Figure 3). These materials can be found at `https://research.cs.wisc.edu/mist/SoftwareSecurityCourse/`.

Educational Components

The basic educational products that we have produced are a comprehensive set of learning units, where each unit will contain the following educational components:

EC1. A video lecture of approximately 10-20 minutes long: Each lecture starts with motivation and learning goals, followed by a sequence of technical topics to develop the student's understanding of the material.

EC2. Closed captioning for each lecture, in both English and Spanish: As time permits and outside resources allow, we will include other languages.

EC3. Lecture slides for instructors: The slides from each unit are available separately in source form (PowerPoint) for trainers and instructors.

EC4. A text chapter that reinforces the material in that lecture: This written guide provides coverage of the basic lecture material, with increased detail, examples, and exercises. In addition, many chapters contain links to further in-depth readings.

EC5. Active learning exercises: Associated with each unit are in-class exercises that reinforce the material in those units. These are meant to be done collaboratively in small groups and then discusses as a whole class.

EC6. Hands-on exercises: These exercises guide the student in trying out the ideas presented in the video and text. Each exercise is delivered in a container or virtual machine image, with a guide and ready-to-run code. The exercises include those that are suitable as traditional homework assignments and those intended for classroom-based active learning.

In the near future, these exercises will be cloud ready and launchable.

EC7. Evaluation and assessment materials: These materials include those for both self-assessment activities and conventional testing. These materials are the basis for certifying the students and trainees.

Delivery Channels

As we mentioned above, our software security educational materials can be applied to a variety of learning contexts. Some of the delivery channels that we address include:

S1. The semester-long class, based on a flipped (blended learning) [5], classroom with active learning exercises: Taken in total, the material that we have developed forms a coherent body of knowledge to support an advanced undergraduate class, such as the Introduction to Software Security (CS542) advanced undergraduate class that we introduced at the University of Wisconsin-Madison and Seguridad y Vulnerabilidad del Software at the Universitat Autònoma de Barcelona. The online lectures, text, and exercises allow the instructor to use class time for discussion, active learning sessions, and evaluation to support each unit.

S2. Supplemental material to support traditional computer science courses, such as courses in operating systems, databases, computer security, or even introductory programming: For example, (1) the SQL Injection unit can used in a database course, (2) the pointers and memory unit in an operating systems course, (3) a selected set of units to provide the software security sections of a broader introductory course on software security, and (4) the unit on introduction to software assurance tools in the introductory programming sequence.

S3. Professional training courses (tutorials) intended for workforce development: Such courses are typically taught in anything from half-day to three-day formats. These courses combine a coherent sequence of video lectures to cover focused topics of interest to the venue (company or organization) and audience. The longer format classes typically include hands-on exercises.

S4. Motivational lecture: There are a variety of venues for a basic introduction to software security, that is accessible to students with the most basic programming background. Examples of such venues include a high school AP Computer Science course, an Hour of Code (https://hourofcode.com/) presentation, or a Hackathon. Such lectures can motivate students to select career paths that lead them into a CI career.

S5. Live lecture class: The presentation materials, separate from the video lectures, can be used by instructors in any of the above contexts to supplement their live lecture materials.

CURRICULUM

We have divided the subject into technical area modules, where each module is divided into units (see Table 1: **Overview of Software Security Curriculum**). Each unit can have a video lecture, text chapter, hands-on exercises, and evaluation activity.

To date, we have 37 video units (with over 10,000 views), 20 text chapters, and 20 hands-on (homework) exercises. In addition, we have 25 active learning exercises and 15 quizzes to support instructor-led class teaching.

This curriculum is a starting point and will evolve based on student, colleague, and organizational feedback, and on our experiences as we introduce new materials to the various learning contexts. As we complete the initial coverage of topics, we will then review existing topics for updates and add new topics. By making our materials free and open, we hope to form a software security community where, the topics will naturally expand as our colleagues will contribute material of their own. For example, Prof. Daphne Yao's group at Virginia Tech is currently developing a unit on the safe use of cryptographic libraries.

Table 1: Overview of Software Security Curriculum

Module	Topic
Module 0	Introduction and welcome
Module 1	Basic concepts and terminology: Provide a common language and background for a precise discussion of software security. Introduce to terms and concepts such as risk, threat, weakness, vulnerability, exploit, confidentiality, integrity, and availability.
	Thinking like an attacker: Introduction to how the experienced attacker views a system and the concept of "owning the

bits”. Redefine exploit from the attacker’s point of view and introduce terms and concepts such as attack surface and impact surface. Describe some classic attacks from this point of view.

Module 2 Thinking like a designer:
The secure design principles, software security life cycle, Microsoft Threat Modeling, and other approaches to incorporating security into program design. Introduction to the security life cycle, trust boundaries, threat identification diagraming, validation, and mitigation. Discuss common threat categories based on the STRIDE model: Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevated privilege. Learn how to capture the design, evaluate the risk, and mitigate the risk.

Module 3 Thinking like a programmer, secure programming:
This module covers a wide variety of coding weaknesses that can lead to vulnerabilities, including pointers and memory, numeric errors (a seriously underestimated category), race conditions, exceptions, serialization, privilege de-escalation, sandboxing, DNS, injections (SQL, command, language, and XML), web (cross site techniques, session hijacking, open redirect), and mobile.

Module 4 Thinking like a systems person, defensive techniques:
This module covers a variety of compiler, operating system, and processor defensive techniques, including address space layout randomization (ASLR), stack canaries, dynamic memory checks, and W⊕X. These techniques make your program more difficult to attack

Module 5 Thinking like an analyst, in-depth vulnerability assessment:
Working as an analyst, learn how to evaluate a software system for vulnerabilities. Vulnerability assessment includes identifying the trust boundaries and attack surface, key software architectural features and resources, trust and privilege analysis, detailed component analysis, and results dissemination. The goal is to focus on the high value assets in a system.

Module 6 Thinking like an analyst, automated assessment tools:
Automated analysis tools, including static and dependency checking, are a base-line technique to be used by every program. Topics for static analysis tools include understanding the conceptual basis for code analyses, including the basics of program control and dataflow, flow sensitivity, context sensitivity, inter-procedural analysis, and pointer analysis. Topics for dynamic analysis tools include program instrumentation and control, code coverage, and input-set generation. Includes a discussion of the practical application of a variety of software assurance tools and tool environments and the limitations of these tools.

Module 7 Dynamic Techniques, fuzz testing and other checkers:
Introduction to fuzz testing as a state-space exploration exercise (an application of “owning the bits” from Module 1.2). Topics include the background and principle of fuzz random testing, techniques for using fuzz tools (such as American Fuzzy Lop), techniques for input generation, and how to develop their own custom fuzz tool.

THE CURRENT SOFTWARE SECURITY CURRICULUM ECOSYSTEM

It is good to see Software Security included as one of eight knowledge areas included in the Cybersecurity Curricula 2017 guidelines produced by a Joint Task Force on Cybersecurity Education [7] and again in the Cybersecurity Curricular Guidance for Associate-Degree Programs [8].

Many computer science programs include undergraduate classes in computer or information security. These courses cover a wide spectrum of topics but focus only in a limited way on software security issues. While these courses have a clear value in developing skills to mitigate cybersecurity risk, they often do not have an emphasis on the software security skills and practices needed to build security into the devices on which we rely within every sector of our economy and for many activities in our daily lives.

A course titled “Build It, Break It, Fix It” originally taught at several U.S. universities became a contest [9] aimed at assessing the ability to securely build software, not just break it. The Software Engineering Institute at Carnegie Mellon University has developed software assurance reference curricula for Masters, Undergraduate, and Community College as well as for Executives.

There are also open resources for learning some software security topics. Of the resources found at the site Free and Low Cost Online Cybersecurity Learning Content maintained by NIST's National Initiative for Cybersecurity Education [6] only one, at the time this article was published, was focused software security. There are also courses from safecode.org. These are short, introductory video modules on a variety of software assurance topics, such as Secure Memory Handling and Cross Site Scripting. While their materials are well prepared, they cover only a few specific areas and are not of sufficient depth and coverage to be used in a university or professional setting, and do not include exercises.

Professional training companies, such as SANS Institute, Secure Coding Academy, Infosec Institute, AppSec Labs, Denim Group, and John Bryce Training College, provide software security training at a cost to the students (or organizations that contract for their use) and are unlikely to share for their materials with other instructors for free use.

HOW WE GOT HERE

Our curriculum was not created in isolation. It grew out of our years of experience doing in-depth software vulnerability assessments and our research efforts to improve and automate the assessment process [10].

Our academic software assessment activities started in 2006 with a request to help increase the security assurance of the infrastructure software that was running in the TeraGrid (the predecessor of XSEDE). Since we were located in the same department as the Condor (now HTCondor) project, and since Condor was a core element of the TeraGrid environment, we started assessing that software. At that time, there were no well-defined processes for approaching such a large body of software for assessment. As part of our assessment activity, we worked to structure our activity and, as a result, developed the First Principles Vulnerability Assessment (FPVA) methodology [11].

Over the years, we have refined FPVA and have had it applied to many important code bases by us and teams that we have trained. Under DHS funding, we assessed Wireshark and Google Chrome. Under NATO and the European Commission funding, we assessed critical Grid software, including MyProxy, VOMS Admin, VOMS Core, glExec, Argus, WMS, and CREAM. In addition, we have recently assessed the software used to control about half of the world's container shipping ports [12], finding and helping to fix major software vulnerabilities that would have allowed an attacker to cause great harm. In addition, we used these experiences to help evaluate the effectiveness of software analysis tools [13].

WHERE DO WE GO FROM HERE?

The materials we have developed are a work in progress. The security field changes rapidly, so we will continue to evolve our materials, updating topics, adding new topics (including those from colleagues), and increasing our reach with closed captions in additional languages.

And, of course, we will continue to proselytize software security in academia, industry, and government. The recent White House Executive Order [14] directly addresses software security and may act to increase interest in software security education and training. As a closing note, our training materials cover attacks on a logging service and command injections, so the knowledge to prevent serious global software vulnerabilities such as the one that enabled the recent Apache log4j2 logging service [15] attack, is already at hand.

ACKNOWLEDGMENT

This paper is a product of Trusted CI. Trusted CI is supported by the National Science Foundation under Grant #1920430. For more information about Trusted CI, visit: http://trustedci.org/. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. We are grate for to Bill Newhouse, Jelena Mirkovic and Blair Taylor for their many useful suggestions to help improve this paper.

■ REFERENCES

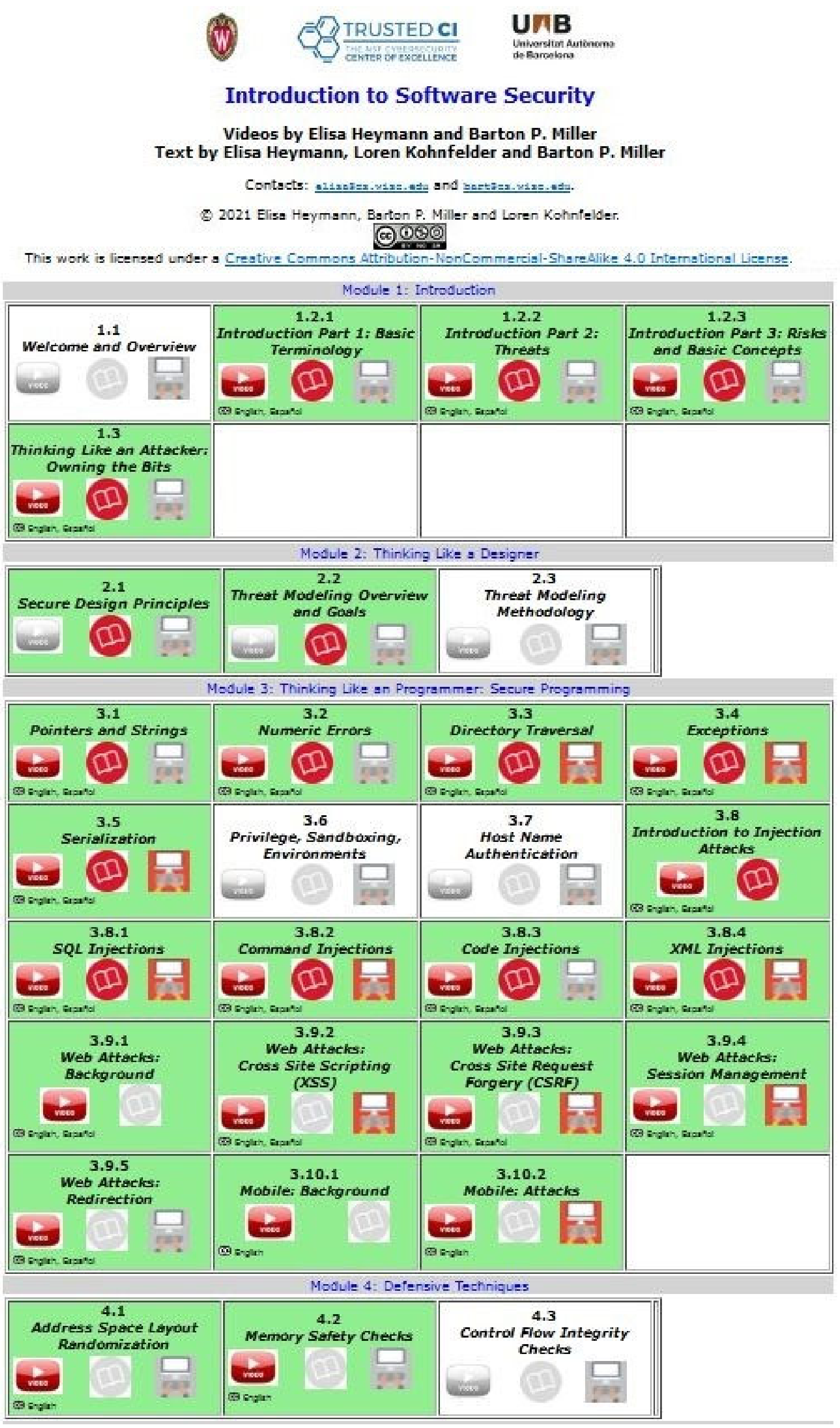
1. Matt Bishop, Deborah Frincke, "Teaching Secure Programming", IEEE Security & Privacy, vol. 3, September/October 2005.
2. Siddharth Kaza, Blair Taylor, Kyle Sherbert, "Hello, World! – Code Responsibly", IEEE Security & Privacy, vol 16, January/February 2018. DOI 10.1109/MSP.2018.1331035.
3. Dave Gruber, "Modern Application Development Security", Enterprise Strategy Group, August 2020. https://info.veracode.com/survey-report-esg-modern-application-development-security.html
4. Stuart Zweben and Betsy Bizot, "2020 Taulbee Survey, Bachelor's and Doctoral Degree Production",

Computing Research Association, 2021. https://cra.org/resources/taulbee-survey/

5. Eric Mazur, Peer Instruction: A User's Manual Series in Educational Innovation, Prentice Hall, Upper Saddle River, NJ, 1997.
6. "Free and Low Cost Online Cybersecurity Learning Content", Information Technology Laboratory, National Institutes of Standards and Technology, https://www.nist.gov/itl/applied-cybersecurity/nice/resources/online-learning-content, accessed December 2021.
7. "Cybersecurity Curricula 2017", Computing Curricula Series Joint Task Force on Cybersecurity Education, December 2017. https://cybered.hosting.acm.org/wp-content/uploads/2018/02/newcover_csec2017.pdf
8. "Cybersecurity Curricular Guidance for Associate-Degree Programs", Committee for Computing Education in Community Colleges, January 2020. https://ccecc.acm.org/files/publications/Cyber2yr2020.pdf
9. James Parker, Michael Hicks, Andrew Ruef, Michelle L. Mazurek, Dave Levin, Piotr Mardziel and Kelsey R. Fulton, "Build It, Break it, Fix It, Contesting Secure Development", *ACM Transactions on Privacy and Security* **23**, 2, article 10, April 2020. https://dl.acm.org/doi/fullHtml/10.1145/3383773.
10. Wenbin Fang, James A. Kupsch, and Barton P. Miller, "Automated Tracing and Visualization of Software Security Structure and Properties", Symposium on Visualization for Cyber Security (VizSec), Seattle, October 15, 2012.
11. James A. Kupsch, Barton P. Miller, Eduardo César, and Elisa Heymann, "First Principles Vulnerability Assessment", 2010 ACM Cloud Computing Security Workshop, Chicago, IL, October 2010.
12. Joseph O. Eichenhofer, Elisa R. Heymann, Barton P. Miller and Kyung Won (Arnold) Kang, "An In-Depth Security Assessment of Maritime Container Terminal Software Systems", IEEE Access, vol. 8, July 2020. https://doi.org/10.1109/ACCESS.2020.3008395. Appeared in earlier form in: Joseph O. Eichenhofer, Elisa Heymann and Barton P. Miller, "In-Depth Software Vulnerability Assessment of Container Terminal Systems", 2nd NATO Conference on Cyber Security in the Maritime Domain, Souda, Crete, Greece, September 2017.
13. James A. Kupsch, Elisa Heymann, Barton P. Miller, and Vamshi Basupalli, "Bad and Good News about Using Software Assurance Tools", *Software: Practice & Experience,* April 2016. http://onlinelibrary.wiley.com/doi/10.1002/spe.2401/full
14. The "Executive Order on Improving the Nation's Cybersecurity", The White House, May 12, 2021. https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/
15. "Apache Log4j Vulnerability Guidance", Cybersecurity & Infrastructure Security Agency, U.S. Department of Homeland Security, December 2021, https://www.cisa.gov/uscert/apache-log4j-vulnerability-guidance

Elisa Heymann is a Senior Scientist on the NSF Cybersecurity Center of Excellence at the University of Wisconsin-Madison, and an Associate Professor at the Autonomous University of Barcelona. She co-directs the MIST software vulnerability assessment at the Autonomous University of Barcelona, Spain. She coordinates the in-depth vulnerability assessments for NSF Trusted CI and was also in charge of the Grid/Cloud security group at the UAB. Heymann participated in two major Grid European Projects: EGI-InSPIRE and European Middleware Initiative (EMI). Heymann's research interests include security and resource management for Grid and Cloud environments. Her research is supported by the NSF, Spanish government, the European Commission, and NATO. Contact her at elisa@cs.wisc.edu

Barton Miller is the Vilas Distinguished Achievement Professor and the Amar & Belinder Sohi Professor in Computer Sciences at the University of Wisconsin-Madison. He is a co-PI on the Trusted CI NSF Cybersecurity Center of Excellence, where he leads the software assurance effort and leads the Paradyn Tools project, which is investigating performance and instrumentation technologies for parallel and distributed applications and systems. His research interests include software security, in-depth vulnerability assessment, binary and malicious code analysis and instrumentation, extreme scale systems, and parallel and distributed program measurement and debugging. In 1988, Miller founded the field of Fuzz random software testing, which is the foundation of many security and software engineering disciplines. In 1992, Miller (working with his then-student Prof. Jeffrey Hollingsworth) founded the field of dynamic binary code instrumentation and coined the term "dynamic instrumentation". Miller is a Senior Member of the IEEE and a Fellow of the ACM. Contact him at bart@cs.wisc.edu



Introduction to Software Security

Videos by Elisa Heymann and Barton P. Miller
Text by Elisa Heymann, Loren Kohnfelder and Barton P. Miller

Contacts: elisa@cs.wisc.edu and bart@cs.wisc.edu.

Module 1: Introduction

1.1 *Welcome and Overview*	1.2.1 *Introduction Part 1: Basic Terminology* English, Español	1.2.2 *Introduction Part 2: Threats* English, Español	1.2.3 *Introduction Part 3: Risks and Basic Concepts* English, Español
1.3 *Thinking Like an Attacker: Owning the Bits* English, Español			

Module 2: Thinking Like a Designer

2.1 *Secure Design Principles*	2.2 *Threat Modeling Overview and Goals*	2.3 *Threat Modeling Methodology*

Module 3: Thinking Like an Programmer: Secure Programming

3.1 *Pointers and Strings* English, Español	3.2 *Numeric Errors* English, Español	3.3 *Directory Traversal* English, Español	3.4 *Exceptions* English, Español
3.5 *Serialization* English, Español	3.6 *Privilege, Sandboxing, Environments*	3.7 *Host Name Authentication*	3.8 *Introduction to Injection Attacks* English, Español
3.8.1 *SQL Injections* English, Español	3.8.2 *Command Injections* English, Español	3.8.3 *Code Injections* English, Español	3.8.4 *XML Injections* English, Español
3.9.1 *Web Attacks: Background* English, Español	3.9.2 *Web Attacks: Cross Site Scripting (XSS)* English, Español	3.9.3 *Web Attacks: Cross Site Request Forgery (CSRF)* English, Español	3.9.4 *Web Attacks: Session Management* English, Español
3.9.5 *Web Attacks: Redirection* English, Español	3.10.1 *Mobile: Background* English	3.10.2 *Mobile: Attacks* English	

Module 4: Defensive Techniques

4.1 *Address Space Layout Randomization* English	4.2 *Memory Safety Checks* English	4.3 *Control Flow Integrity Checks*

Figure 1: **Current Online Instructional Material Web Site**
`https://research.cs.wisc.edu/mist/SoftwareSecurityCourse/`